WordPressプラグインを作ったあとに放置しないために

テスト自動化と公式ディレクトリへの公開自動化

2026-07-25 / 信州WordPress Meetup in 松本 Vol.40

自己紹介

- ID ytsuyuzaki / 露崎勇大 (つゆざき ゆうだい)
- Webプログラマー / フリーランス
- PHP使いの野鳥好き
- 信州WordPress Meetupでは何回か発表しました
  - 前回: WordPressの情報から返答するチャットbotを作ってみた
  - 過去: WordPress診断のWebサービスMozCheckの紹介
- 直近で取り組んでいるトピック
  - WordPressで複数サイト運営してAdSenseをガチる
  - LaravelでWordPress診断/監視サービスのSaaS開発
- 今回からイベントホストに選ばれました

今日話すこと

- これから WordPress 開発はどう変わると感じているか
- その中で、開発したプラグイン/テーマの保守に何が必要か
- 今回、何を「できる状態」にしたのか
- 保守の自動化をしてみて分かった怖さと手応え

話さないこと

- 具体的な設定方法（別途ブログ記事にまとめています）

きっかけ

年末に、納品したLaravelアプリで使っていたlivewireライブラリ
の脆弱性被害に遭った。

作ったものを公開したまま、放置する怖さをかなり強く感じた。

それから半年間、既存の制作物の総点検を開始

- Laravelで作った公開中/配布中のWebアプリケーション
- Electronで作ったデスクトップアプリ
- Vimプラグイン
- Composerで配布しているパッケージ
- WordPressプラグイン

などなど

ソフトウェアは更新が不要、という誤解

(私以外にも思い当たる人がいるはず...?!)

- 家電や本のように、一度完成したらそのまま使い続けられるという感覚がある
- Webサイトを資産ではなく制作物として見ている
- セキュリティ修正や内部改善は画面上の変化がなく、お金が発生しない
- 「アップデートしたら壊れた」という話は印象に残り、
  「アップデートしたから攻撃を防げた」は知ることが出来ない

ソフトウェアは変わらずとも周りの環境が変わるので、変えていく必要がある。

技術革新でのセキュリティリスクやメンテされなくなった物からの乗り換え等々

ここまでは自分の話。
次からはWordPressに当てはめた話。

WordPressフォーラムで話題にあがるよくある話

- 更新に失敗した
- 更新しなかったせいでサイトが改ざんされた
- 更新したら画面がまっしろになった
- 今まで出来ていた事ができなくなった

WordPressを使う上で、常に最新版にするのが推奨されているが、
更新できるように作るのがそれなりに労力がかかるし、
そこにお金は出てこない事が多い。

更新が必要なのに、更新できるようにはなりづらい

さらに、これから増えると思っているもの

- 生成AIでコードを書くことが気軽になった
- 生成AI製の小さなプラグインが増える
  - 個人や小規模チームが作る業務用プラグイン
  - 公式ディレクトリに出ない独自配布のプラグイン
  - 俺が考えた最強のWordPressテーマ
- 作った本人も細部を覚えていないコード

作るハードルは下がる。
保守のハードルは(まだ)下がっていない。
これからだんだんと保守についてのトピックが出てくるはず (おそらく(期待))

問題意識

作れる人が増えるほど、

「直せる状態」「更新できる状態」が重要になる。

われわれ開発者が見るべき変化する環境

- WordPress 本体が更新される
- PHP が更新される
- npm / Composer で依存しているプラグインが更新される
- ブロックエディタの前提が変わる
- 使い勝手や期待されるユーザービリティの変化
- セキュリティ問題があとから出る

プラグイン/テーマは公開した瞬間に終わりではなく、そこから変化が始まる。

経験や未来予測から、目指したもの

「完璧なテスト環境を作る」ではなく、

- 変更しやすい環境
- 問題に気づける状態
- 公開作業を怖がりすぎない仕組み

を作ること。

WordPressプラグイン「BeastFeedbacks」を例に解説

WordPressプラグイン「BeastFeedbacks」とは

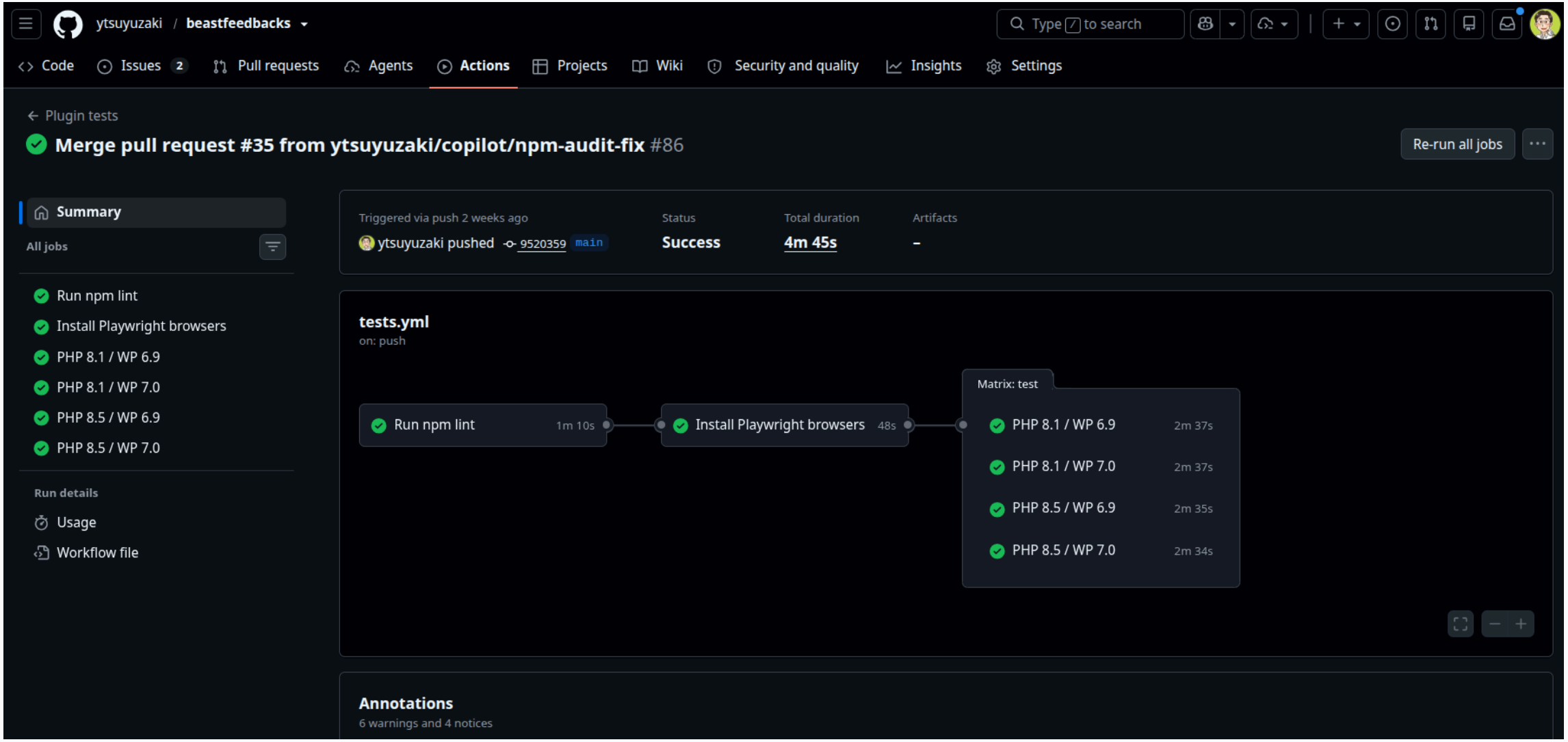
- 公式ディレクトリからダウンロードが可能
- ユーザーフィードバックを取得するためのブロック機能を提供
  - アンケートフォーム
  - いいねボタン
  - 選択投票 (この記事は有益でしたか？ YES/NO)
- WordPressで収集できる一般的なアクセスログ、コメント、から一歩進んだ「ユーザーが何を考えているのか」を収集することができる
- 競合プラグイン
  - Crowdsignal フォーム（Automattic）
  - Contact Form 7（Rock Lobster Inc.）

BeastFeedbacksでやったこと

脆弱性の発見、テストで確認、公開の自動化

- Dependabot で使用ライブラリの脆弱性検知
- GitHub Actions でLintやテストの実行
  - wp-scripts (同梱されているlint系一通り)
  - wp-env でPHPやWordPressのバージョン指定で環境立ち上げ
  - PHPUnit、Playwrightでのテスト実行
- タグ作成
  - WordPress.org SVN へのデプロイで更新を公開

実際の設定方法や内容については割愛
詳細は 別途ブログ記事 にまとめています。


ytsuyuzaki / beastfeedbacks
Type / to search
Code
Issues 2
Pull requests
Agents
Actions
Projects
Wiki
Security and quality
Insights
Settings
Plugin tests
Merge pull request #35 from ytsuyuzaki/copilot/npm-audit-fix #86
Re-run all jobs
Summary
All jobs
Run npm lint
Install Playwright browsers
PHP 8.1 / WP 6.9
PHP 8.1 / WP 7.0
PHP 8.5 / WP 6.9
PHP 8.5 / WP 7.0
Run details
Usage
Workflow file
Triggered via push 2 weeks ago
ytsuyuzaki pushed 9520359 main
Status
Success
Total duration
4m 45s
Artifacts
–
tests.yml
on: push
Run npm lint 1m 10s
Install Playwright browsers 48s
Matrix: test
PHP 8.1 / WP 6.9 2m 37s
PHP 8.1 / WP 7.0 2m 37s
PHP 8.5 / WP 6.9 2m 35s
PHP 8.5 / WP 7.0 2m 34s
Annotations
6 warnings and 4 notices

今回できたことの要約

1. 問題に気づけるようになった

依存パッケージの脆弱性や更新を、GitHub 上で見つけられるようにした。

以前は「見に行かないと分からない」状態だった。

今は「問題が来たら通知される」状態になった。

2. 変更を気軽に試せるようになった

WordPress を起動し、対象プラグインを有効化し、実際に近い環境で確認。
これをコマンド一発 or 自動化できるようにした。

PHP の処理だけでなく、ブロックエディタ上のブラウザ操作も確認対象にした。

3. 人間の記憶に頼る部分を減らした

確認項目を毎回思い出すのではなく、push や pull request をきっかけに自動で確認する。

これで「何を確認すればいいんだっけ」という迷いが減った。

4. 公開作業を手順化できた

タグを作ると、リリース用の zip を作り、WordPress.org への反映まで進められるようにした。

公開作業が特別な儀式ではなく、繰り返せる手順になった。

達成できたこと

「直すのが面倒だから放置する」から、

「直して確認して更新できる」へ近づいた。

ちなみに、やってみて失敗したところ

公開の自動化で失敗

WordPressプラグインの公開設定、
SVNリポジトリへのコミットが必要なのですが、zipファイルからの解答や階層設定のミスが判明。
必要な設定が足りていない状態で公開され、一時的に画面が真っ白になった。

自動化は、正しい作業だけでなく、間違った作業も速く反映してしまう。
（公開されたばかりで利用者が居なかったのが救いだった）

そこからの学び

- いきなり重要なものに入れない
- 影響範囲の小さいものから試す
- 公開前に見る場所を決めておく
- 自動化した後も、最後の確認は設計する

便利さより先に、失敗したときの戻り方を考える。

未来の話

生成AIで、作る量は増える

プラグインや小さな機能は、これまでより簡単に作れるようになる。

ただ、そのぶん「作られたけれど保守されないもの」も増えると思っている。

だから保守の型が必要になる

これから大事になるのは、すごいコードを書くことだけではなくて

- 問題に気づける
- 変更を試せる
- 失敗を検知できる
- 必要なときに公開できる

この型を持っていることが、開発の継続力になる。

宣伝：継続的な診断/監視ができる MozCheck

WordPress の不安が解る診断サービス。
URLの入力だけでパフォーマンス、セキュリティ、
SEOの状態を定期的にチェックします。

https://mozcheck.jp/

無料ユーザーで3サイト分の定期診断に対応。
将来的に監視機能として以下を対応予定

- ステータスコードのチェック
- 表示確認
- 問い合わせフォームの送信チェック

まとめ

これは単なるテストを設定したという話ではありません。
（WordPressプラグイン、Hello Dolly風の言い回し）

WordPress制作を、作ったあとも直し続けられる状態に近づける取り組みです。

ありがとうございました

元記事:

WordPressプラグインのアップデート、保守のためのテスト自動化について